Chapter of the book "Knowledge Engineering and Agent Technology"
J. Cuena et al. (Eds), IOS Press, 2004

Using Knowledge Modelling Tools for Agent-based Systems: The Experience of KSM

Martin MOLINA and Jose CUENA
Department of Artificial Intelligence Technical University of Madrid,
Campus de Montegancedo s/n, 28660 Boadilla del Monte (Madrid), Spain
http://www.dia.fi.upm.es

Abstract. The aim of this chapter is to discuss the applicability of recently proposed knowledge modelling tools to the development of agent-based systems. The discussion is derived from the real world experience of a particular software tool called KSM (Knowledge Structure Manager). The chapter provides details about this tool and then proceeds to show in which forms the software may be used to support the development of agent-based systems. Two multiagent systems, one in the field of telecommunications management and the other one in the field of flood control, are described. Conclusions about these studies are presented, summarizing the main contributions that knowledge modelling tools can bring to the development of agent-based systems.

1. Introduction

Consideration of computer systems as agent-based systems has received increasing attention as design paradigm. One distinguishes between two different types of multiagent architectures: *(1) complex societies of simple, homogeneous agents*, wherein the system complexity is derived from the existence of a large number of agents of quite simple internal architecture; and *(2) simple societies of complex, heterogeneous agents*, wherein each agent possesses a significant internal architecture and where the society usually encompasses different types of agents. With respect to the second type, knowledge-based techniques can be appropriate solutions for construction of the individual agents.

In this second type of software architectures, the usual internal knowledge complexity of each agent, that determines both individual and social behavior, requires an adequate process for modulating and configuring of the cognitive capacities of the individual agent. For this purpose, it is convenient to make use of recent advances in the field of knowledge engineering involving methodologies and tools that can provide criteria and technical solutions to cope with the usual complexity and diversity of knowledge found in each agent. Thus, it is very adequate to combine agent-based technology with knowledge-based technology to facilitate the development of more complex systems in real world problems.

According to this, the present chapter describes the use of an advanced knowledge engineering tool in the development of agent-based systems. The chapter describes first the characteristics of a particular knowledge engineering software environment: the KSM tool. Subsequently, a general approach is shown for using this tool in the development of agent-based systems. This is done with the discussion of two case studies corresponding to two different real world systems: one in the field of telecommunications management and the other one in the field of flood control. Finally, general conclusions derived from these studies are presented.

2. The KSM tool for knowledge modelling

This section summarizes the main characteristics of the KSM tool, a software environment that was designed to help developers and end-users in the development and maintenance of large and complex knowledge-based systems following a modelling approach. The section includes a first part that describes the main knowledge modelling concepts followed by KSM. Then, the second part presents the types of symbolic knowledge representation languages provided by KSM. Finally, this section describes the characteristics of the KSM software environment. More detailed descriptions about this tool can be found at [1], [2], [3], [4], [5].

2.1. Knowledge modelling concepts

In order to characterize the knowledge model of an agent it is possible to apply the paradigm of *model-based* system development, which has become a popular approach to the development of complex knowledge-based systems. This modelling approach considers the existence of an abstract level (proposed by Newell with the name of *knowledge level* [6]) at which the knowledge can functionally described on the basis of its role, independently on the particular symbolic representation (this view contrasts to the traditional approach where a knowledge system was usually considered as a container to be filled with knowledge extracted from an expert). Some recent methodologies for system development such as KSM and others (KADS [7] or Protégé-II [8]) follow this model-based approach.

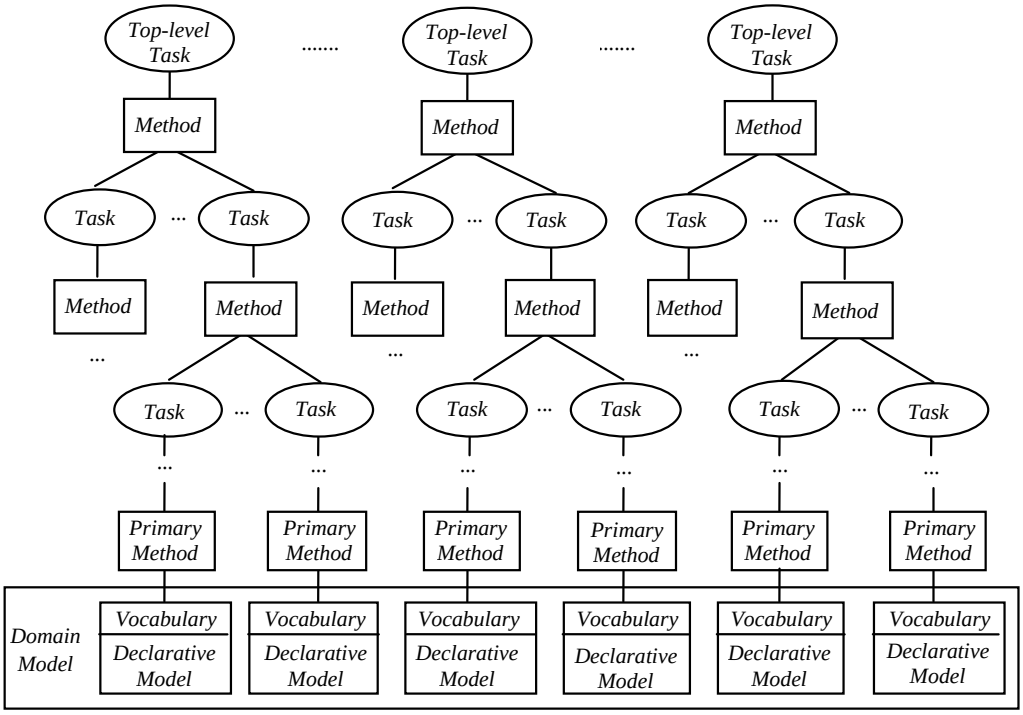

Figure 1: Hierachies of task-method-domain structures to describe a knowledge-model

These methodologies organise the knowledge according to certain structuring principles. One organisation followed by most of the methodologies is the task-oriented approach. According to this view, a *task* is an abstract description that identifies a goal to be achieved (for instance, mineral classification or the design of the machinery of an elevator). Tasks are usually characterised by the classes of premises that they receive as input and the classes of conclusions that they produce as output. On the other hand, *problem-solving methods* (or *methods* in short) are used to cope with the tasks. A method

indicates how a task is achieved, by describing the different reasoning steps by which its inputs are transformed into outputs. Simple tasks can be attained directly by means of primary methods. What is considered as a primary method is a design decision established by the developer. Typically, primary methods correspond to methods that can be directly implemented at symbolic level by simple problem-solving techniques (such as knowledge based techniques like backward or forward chaining in rule-based representations, constraint satisfaction methods, and also specific algorithmic solutions that do not require a explicit representation of declarative knowledge). Primary methods rely on a knowledge base, modelling the declarative domain knowledge used by basic methods. The use of declarative knowledge by primary methods requires an ontological definition of such a knowledge that is viewed as a set of *domain models* that support primary tasks. This type of description based on tasks and methods was originally present in several proposals from different authors such as the *generic task* [9], [10], the KADS model [7], the *components of expertise* [11], the *role limiting method* [12]. Following this approach, a model can be described initally as a collection of *top-level tasks* that identify the set of main goals to be achieved by the application. These tasks requires compound methods that decompose them into subtasks. These subtasks may again be decomposed by a method and so on, developing a task-method-domain hierarchy (figure 1), whose leaves are given by basic methods that use simple knowledge bases. Thus, the resulting model for an agent can be viewed as a collection of types of knowledge bases (each one with its own symbolic representation) together with a hierarchically structured set of reasoning strategies that make use of such knowledge bases.

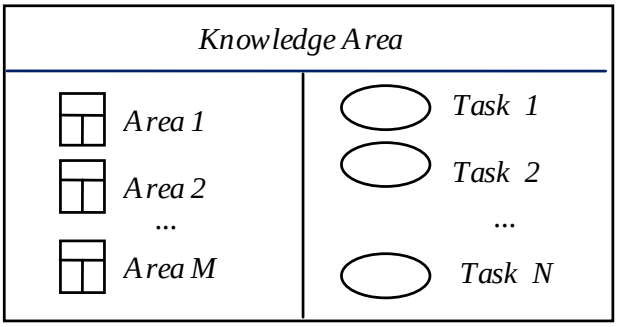

Figure 2: The structure of a knowledge area

However, in real applications, the experience shows that, sometimes, too large descriptions can be designed by using this type of formulation that may produce problems of understanding and maintainance together with problems of efficiency in the final software implementations. Thus, although the conceptual description based on task-methods-domains is adequate for the analysis process, it needs to be complemented and re-organized using additional modelling concepts for the final design of the application, to have a synthetic view of an application at several levels of conceptual aggregation. For this purpose, an additional description entity can be used. This entity is called *knowledge area* and was originally proposed as a main structuring concept within the KSM tool [1], [2].

A knowledge area in KSM follows the intuition of a body of knowledge that explains a certain problem solving behaviour observed in an agent. A cognitive architecture that models the expertise can be viewed as a hierarchically structured collection of knowledge areas at different levels of detail, where each knowledge area represents a particular qualification or speciality that supports particular problem solving actions. A knowledge area (figure 2) is described with two parts: (1) its knowledge, represented as a set of component sub-areas of knowledge, and (2) its functionality, represented by a set of tasks (and their corresponding methods). The first part decomposes the knowledge area into simpler subareas, developing a hierarchy at different degrees of detail. The second part associates tasks to knowledge areas showing their functional capabilities.

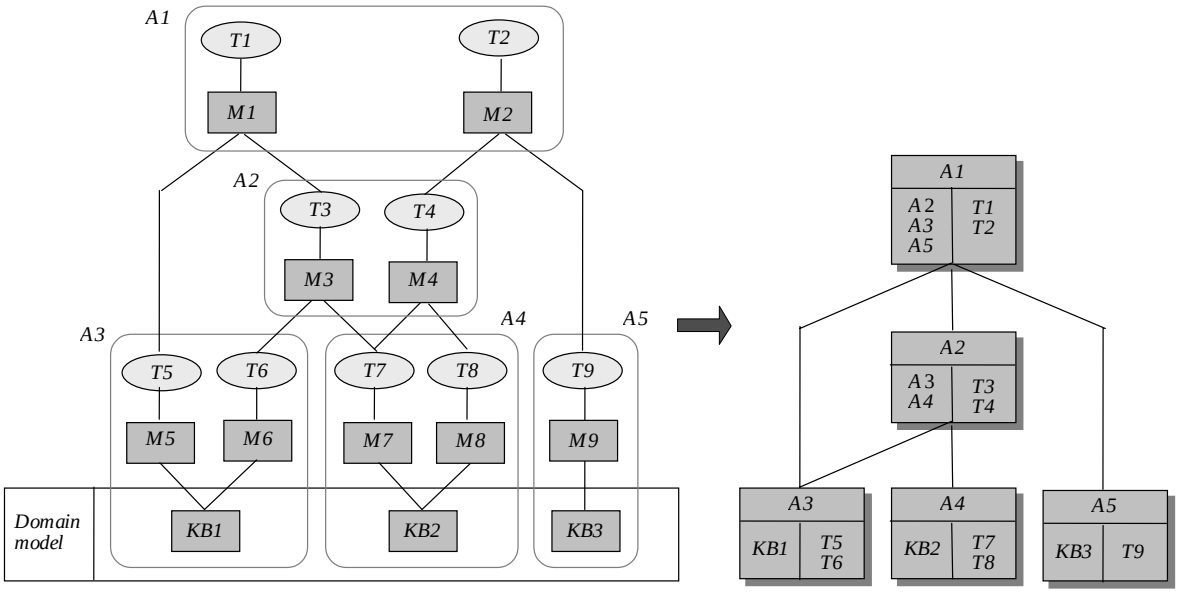

Figure 3: Example of the encapsulation provided by the knowledge area concept. The hierarchy of task-method-domain on the left can be grouped using five knowledge areas on the right.

The knowledge area concept is useful to produce a more synthetic view of the knowledge model given that it groups a set of tasks (together with the corresponding methods) in a conceptual entity of higher level. Figure 3 shows how the tasks of the task-method-domain hierarchy on the left can be grouped in a hierarchy of five knowledge areas on the right. In principle, given a hierarchy of task-method-domain resulting from the knowledge level analysis, it is possible to design different hierarchies of areas according to a principle of knowledge area structuring. In order to guarantee a reasonable level of understandability of the knowledge model, the final hierarchy should be designed to follow the natural intuitions associated to the knowledge attibuted to the human problem solving process to be modelled. The formulation using the knowledge area format provides certain advantages: (1) every task at any level of the hierarchy is associated to the explicit knowledge that supports its functionality, which makes more meaningful the model (2) the structure of knowledge areas synthesizes the structure of tasks-method-domains, which is useful to better understand complex models, (3) at the bottom level, primary knowledge areas encapsulate declarative domain models, so it is a solution to organize the domain layer in separate modules, which contributes to keep easier the consistency of the model and (4) primary knowledge areas are easy to be implemented by reusable and efficient software components, which gives a solution for the development and maintenance of the final executable version of the system.

Knowledge areas can be defined at *generic* and at *domain* level. Generic areas mean classes of bodies of knowledge that allow to formulate a model. Then, a particular domain model is viewed as a collection of instances of such classes that can share by inheritance different properties of the classes such as relations with other areas, problem-solving methods, etc. This possibility of defining classes of areas is a solution to support reuse. Thus, abstract structures of knowledge areas may be reused to develop different applications operating in different domains. This structuration contrasts to a plain organization of knowledge, such as the traditional structure proposed in the original rule-based systems that does not describe explicitly the different knowledge modules in which rules could be organized. Knowledge areas allow to identify such modules, even by establishing several conceptual levels (knowledge areas being part of other knowledge areas). Thus, the resulting system can describe better its own knowledge (more similar to how a human expert does) showing the categories in which it can be classified. This contributes to present different levels of detail of the expertise, and to produce explanation of good quality. Section 4 includes specific examples of the use of this methodology in two real-world problems.

2.2. Symbolic knowledge representation in KSM

One of the basic assumptions followed by the KSM approach to develop a knowledge model is that, instead of using a uniform symbolic knowledge representation for the whole model (e.g., logic or rules) that can be useful for the analysis and formalization phases but could be artificial and inneficient for the development of the final application, the developer will use for each case the most appropriate symbolic representation in order to produce both an efficient and a comprehensible model. According to this, KSM distinguishes between three categories of knowledge for which there are different approaches for symbolic knowledge representation: (1) domain *knowledge bases* corresponding to primary areas, (2) *conceptual vocabularies*, i.e., common basic domain terminologies shared by different knowledge bases, and (3) procedural knowledge to formulate *problem-solving methods*. For the first category, KSM provides a library of reusable software components, called *primitives of representation*, that offer the required freedom to the developer to select the most convenient representation for each case (rules, frames, constraints, belief networks, etc.). For the second and third categories, KSM provides two languages the Concel and the Link languages. The rest of the section explains in more detail these three types of components for symbolic representation.

The primitives of representation for knowledge bases

As it was presented, the central structure of a knowledge model is defined as a hierarchy of knowledge areas, where each area is divided into subareas until elementary areas are reached (called *primary* knowledge areas). The purpose of a *primitive of representation* is to provide a symbolic representation together with a set of primitive inference methods to build the operational version of a primary area of a knowledge model (a more detailed description of this type of components can be found at [13]). For each primary knowledge area of the model, the developer selects the most appropriate primitive that acts like a template to be filled using domain knowledge in order to create the final operational component that implements the primary area. The primitives of representation are taken from an library of primitives provided by KSM. This library is open, i.e., new primitives can be included as a result of the development of new software components.

Each primitive of representation is considered as a reusable pre-programmed software component that implements a generic technique for solving certain classes of problems. The primitive defines a particular domain representation using a declarative language together with several inference procedures that provide problem-solving competence. In a simplified way, the structure of the primitive is defined by a pair $<L, I>$, where L is a formal language for knowledge representation and $I = \{ij\}$ is the set of inferences, i.e., a collection of inference procedures that use the declarative representation written in L. The module defined by a primitive is a design decision that is mainly influenced by the representation language L. This language is usually homogeneous, declarative and close to personal intuitions or professional fields. In a particular primitive, this language can adopt one of the representations used in knowledge engineering such as: rules, constraints, frames, logic, uncertainty (fuzzy logic, belief networks, etc.), temporal or spatial representations, etc. Also other parameterised or conventional representations can be considered, such as the parameters of a simulator or a graph-based language. Each element of the set of inferences I expresses an inference procedure that uses the knowledge formulated in the language L. For instance, the rule-based primitive may have an inference, called *forward chaining*, that implements an inference procedure following a forward chaining strategy to determine whether a goal can be deduced from a set of facts given the rules of the knowledge base. In addition, there may be also another inference that follows the backward chaining strategy for the same goal. Each inference ij defines a pair $<P, C>$ where P is a set of inputs (premises) and C is a set of outputs (conclusions).

The primitive provides an interesting level of generality due to the abstraction of the domain knowledge that provides the use of the representation language. The same primitive can be used to construct different modules with different domain knowledge. For instance, a rule-based primitive can be used to construct a module to diagnose infectious diseases or it can be used to build a module that classifies sensor data. Both modules are supported by the same primitive but they include different domain knowledge. Another

interesting advantage provided by the primitive is that there is a clear analogy between primitives and knowledge areas, so this offers an easy transition from the implementation-independent model (as a result of analysis phase) to a more refined model where elementary computable components have been selected to configure the operational version. This continuity preserves the structure defined by the abstract model and, as a consequence, improves the understandability and flexibility of the final system.

The representation language of the primitive is used to formulate a declarative model of the domain knowledge. However, this local information could be shared by other different primitives. This problem about common concepts is solved by using of conceptual vocabularies (see below). From the point of view of primitives of representation, they need to be capable of sharing vocabularies. The solution to this is that the primitive provides mechanisms to import vocabulary definitions that are translated to the local representation language of the primitive. Thus, when the user of the primitive needs to write a particular local knowledge base during the knowledge acquisition phase, the vocabularies shared by the primitive are previously imported to be part of the base, in such a way that vocabularies are directly available in the language of the primitive to help in writing the knowledge base.

At the implementation level, the primitive is a software module designed and implemented as a *class* (from the object-oriented development point of view), i.e. programmed with a hidden data structure and with a collection of operations which are activated when the class receives messages. The language where each primitive must be formulated is open. Different programming languages such as C,C++, Fortran, Prolog, etc may be applied. If they are knowledge-based they must have a user interface to acquire the structures of representation for the knowledge base (such as rules or frames). This activity is carried out by programmers outside of KSM using particular programming languages. Once a particular primitive is built, it must be individually validated and then it is integrated in the KSM library as a reusable software component for building several applications. The executable version of the knowledge model is built by duplicating, adapting and assembly primitives using the KSM facilities.

The Concel language for common terminologies

In order to facilitate an efficient operationalization of the final model, it is important to distinguish between the domain descriptions that are common to the whole model and additional extensions oriented to perform specific primary tasks. In KSM, the common descriptions are formulated with what is called *conceptual vocabularies*. A conceptual vocabulary allows the developer to define a common terminology which can be used by different primary knowledge areas. One of the direct advantages of the use of vocabularies is that they provide a common location where concepts are defined. This avoids to repeatedly define the same concepts eliminating the risk of incoherence in the knowledge of different domains. The concepts defined by the vocabulary will be later referred by other symbolic representations (rules, frames, constraints, etc.) used by primary areas. Due to the general use of vocabularies by different knowledge modules, they must be formulated in a common language. KSM provides the Concel language for this purpose. It allows the developer to define: concepts, attributes, facets and values and the classification of the concepts in classes and instances.

Figure 4 shows an example of the definition of some concepts using the Concel language.This example defines the class called *urban section* as a subclass of the concept *section*. It is defined with six attributes where there are both numerical and qualitative attributes. For instance the attributes *lanes* and *length* are integers (with ranges 1-4 and 0-1000 respectively) and the attribute *capacity* is an interval (with range 0-2000). There is a default value for the attribute lanes (one lane). The attributes *capacity* and *length* have units (vehicles/Km and meters respectively). On the other hand the attributes *detectors*, *speed* and *circulation* have qualitative values. In the case of *speed* and *circulation* they present explicitly the set of possible values (e.g., low, medium and high for speed). The type of values of the attribute detectors are defined as instances of the class detector. In addition to that, the concept *Main Street* is defined as an instance of the class *urban section*. In this case, particular values are associated to some attributes defined in the class. Usually, a generic model include conceptual vocabularies that define classes of concepts

(and possibly also instances) that are domain-independent. The particular instances or subclasses of such concepts corresponding to a specific domain will be defined later when the model is instantiated on such a domain.

```
CONCEPT urban section SUBCLASS OF section.
ATTRIBUTES:
     capacity     (INTERVAL RANGE 0 2000) [veh_Km],
     lanes        (INTEGER RANGE 1 4): 1,
     detectors    (INSTANCES OF Detector),
     length       (INTEGER RANGE 0 1000) [m],
     speed        {low, medium, high},
     circulation  {free, saturated, congested}.
...
...
CONCEPT Main Street IS A urban section.
ATTRIBUTES:
     capacity:    [1400, 1800] [Veh_Km],
     lanes:       3,
     detectors:   (DE1003, DE1005),
     length:      350 [m].
...
...
```

Figure 4: Partial example of concepts definition using the Concel language

The Link language for problem-solving knowledge

On the other hand, in order to describe how a task is carried out, a developer defines a method with a particular problem-solving strategy using the Link language. Methods may be considered control knowledge given that they describe control strategies about the use of domain knowledge Basically, using the Link language, the method formulation is described by two main parts (see example of figure 5): (1) the *data flow section*, to define the data connection among subtasks and (2) the *control flow section*, to formulate the execution order of subtasks. In addition to this, there are also other two optional sections (for reflective behaviour and for default values of search control parameters) which are not described here in detail (for a deeper description of the Link language, see [4]).

The *data flo*w section describes the data connection of subtasks showing how some outputs of a subtask are inputs of other subtasks. The developer here writes input/output specifications of subtasks. Each i/o specification includes (1) the subtask name as a pair made of the knowledge area name and the subtask identifier, (2) the input of the subtask with names identifying data (here, each input flow accepts a *mode* to either get all the elements of a list at once or element by element, which is useful to formulate non-deterministic search methods), (3) the output is defined with a list of single identifiers. In Link language, in general, subtasks are considered non-deterministic processes. This means that as a result of a reasoning, a task may generate not just one result, but several ones. For instance, in the context of medical diagnosis, the task may deduce several diseases and several therapies for the same symptoms. Thus, when tasks are going to be connected in the data flow section this possibility must be taken into account. This is managed with two output modes. Modes select whether the whole set of outputs must be generated one by one element considering that there is a non-deterministic result (this is the default mode) or, on the contrary, it must generate all the outputs at once as a list of single elements for each output flow, which is called the *all* mode.

The purpose of the *control flow* section is to formulate the strategic knowledge that determines the execution order of subtasks. The representation uses production rules. The advantage of this representation is that it easily may define local search spaces considering the non-deterministic behaviour of subtasks. At the same time, the representation is simple enough to be used easily due to this language is not a complex programming language but, on the contrary, it serves as an easy description language to formulate procedural knowledge (a method will have a small number of rules, usually less than 10). Using production rules provides a intuitive representation, and flexibility for maintenance. The format of each rule is: (1) the left hand side includes a set of conditions about intermediate *state of task executions*, and (2) the right hand side includes a sequence of *specification of*

task execution. Each one of the first elements (state of task executions) includes a reference about a knowledge area, a task identifier of that area and a control state, which means that the result of the execution of the task has generated the control state. The representation of the elements in the right hand side (specification of task execution) includes a knowledge area, a task and a execution mode, to indicate a task to be executed with a particular mode.

```
METHOD establish and refine

ARGUMENTS
           INPUT description
           OUTPUT category

DATA FLOW
           (validity) establish
                       INPUT description, hypothesis
                       OUTPUT category
           (taxonomy) refine
                       INPUT category
                       OUTPUT hypothesis

CONTROL FLOW
           START
           ->    (taxonomy) refine, MODE maximum answers=3,
                 (validity) establish.

           (validity) establish IS established,
           (taxonomy) refine IS intermediate hypothesis
           ->    (taxonomy) refine MODE maximum answers=3,
                 (validity) establish.

           (validity) establish IS established,
           (taxonomy) refine IS final hypothesis
           -> END.
```

Figure 5: Example of method formulation using the Link language

For instance, figure 5 shows a complete example of a method formulation for hierarchical classification using the *establish-and-refine* strategy. Besides the global inputs and outputs, the method includes a section for data flow with two tasks and a section for control flow with three rules. The representation with rules includes references to the beginning and the end of the execution to indicate the first set of actions to be done and when the process has reached a solution of the problem. The beginning of the execution is referred as a state of the execution (to be included in the left hand side of the rules) and it is written with the reserved word START. The end of the execution is considered as an action (to be included in the right hand side of the rules) and it is written with the reserved word END.

The execution of a method formulated using the Link language follows the control established by the set of control rules. In the simplest case, when this sequence is permanent, there is just one rule with the explicit order at the right hand side. However, the use of control rules allows to define more complex situations. First, it allows to dynamically determine the sequence of execution, so that it is possible to represent control structures such as *if-then, loops, repeat*, etc. In order to do so, control states are used. For instance, in the previous example of method that follows the establish and refine strategy, the second rule can be triggered in a loop until the hypothesis is not intermediate. In addition to that, in Link language it is possible to define a more powerful execution with a non-linear sequence. This is possible by two reasons: on the one hand, for a given state more than one rule may be used and, on the other hand, a given task may generate more than one result. This possibility of non-linear executions is a powerful technique that allows the developer to define more easily problem-solving strategies where there are search procedures. According to this, Link develops a local search space for the execution of a particular problem-solving method. In general, given that a method calls subtasks, each one with its particular method, different local search spaces are developed at run time by the Link interpreter, each one for each method.

2.3. The KSM software environment

The previous knowledge modelling approach is supported by the KSM sofware environment. The KSM environment helps developers and end-users to construct and maintain large and complex applications, using both knowledge-based and conventional techniques. KSM covers different steps of the life-cycle of an application:

- *Analysis*. KSM uses a particular modeling paradigm, based on the knowledge area concept, for a high level description of the knowledge of the application. The developer uses this paradigm to create a conceptual model to be accepted by the end-user before starting the implementation. Unlike the conventional models of software engineering based on a perspective of information processing, this model is focused on knowledge components which provides a richer and more intuitive description of the architecture of the application. The analysis phase may be either (1) *totally creative*, i.e. the model is only derived from the information provided by domain experts, or (2) *model-based*, i.e., the model is also derived from a generic model taken from a library of reusable models that establishes the abstract structure of components and relations.
- *Design and implementation*. KSM assists the developer to create the final executable version of the knowledge model. In order to do so, KSM manages reusable software components (the primitives of representation) which are adapted and assembled by the developer following the structure of the conceptual model. Normally, primitives provide general inference procedures and representation techniques to write knowledge bases (although also domain dependent primitives can be considered). In this phase it is also required to fill in the architecture with the specificities of the problems to be solved. For this purpose the domain models are to be formulated by introducing parameter values and knowledge bases required for case modeling.
- *Operation and maintenance*. Once the application is built, the end user can apply KSM to consult the structure of the conceptual model of the application and may access to local independent knowledge bases following this structure. The role of KSM here is to allow the end-user to open the application to access to its knowledge structure so that, instead of being a black box like the conventional systems, the final application shows high level comprehensible descriptions of its knowledge. The user also may change the conceptual model at this level, without programming, in order to adapt the system to new requirements. KSM automatically translates these changes into the implementation level.

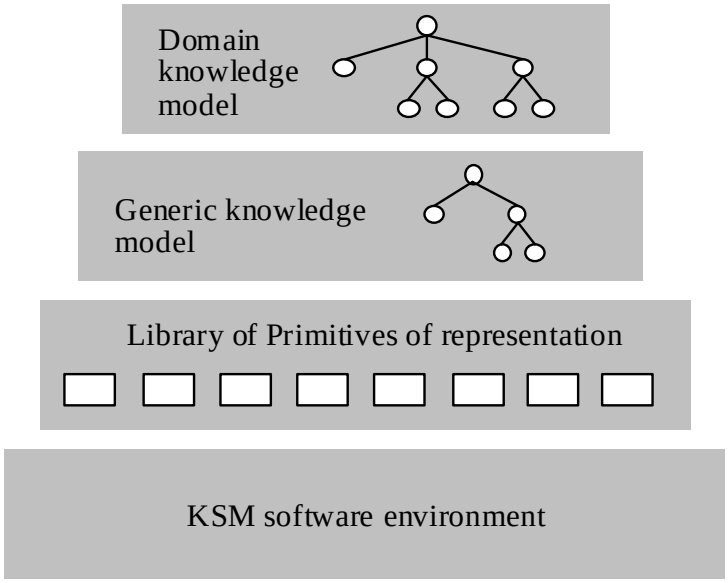

Figure 6: Main building blocks corresponding to a complete knowledge model in KSM. It is considered as a structure of four layers, where each layer is supported by the layer immediately below.

Figure 6 shows the main building blocks that correspond to a complete knowledge model developed using KSM. It is considered as a structure of four layers, where each layer is supported by the layer immediately below. Here the KSM software tool serves as a plataform to support the rest of the components. Immediately above this platform, the library of primitives of representation provide the basic set of software components that support the next layer, the generic knowledge model considered as an abstract knowledge structure. Finally, at the top, the domain knowledge model shares the generic model to organize the specific knowledge bases corresponding to the particular domain. Thus, KSM conceives the final application as a modular architecture made of a structured collection of basic modules. At the implementation level, each elementary module is a reusable software component programmed with an appropriate language and a particular technique (knowledge-based or conventional). Using KSM, a developer can duplicate, adapt and assemble the different software components following a high level knowledge model which offers a global view of the architecture.

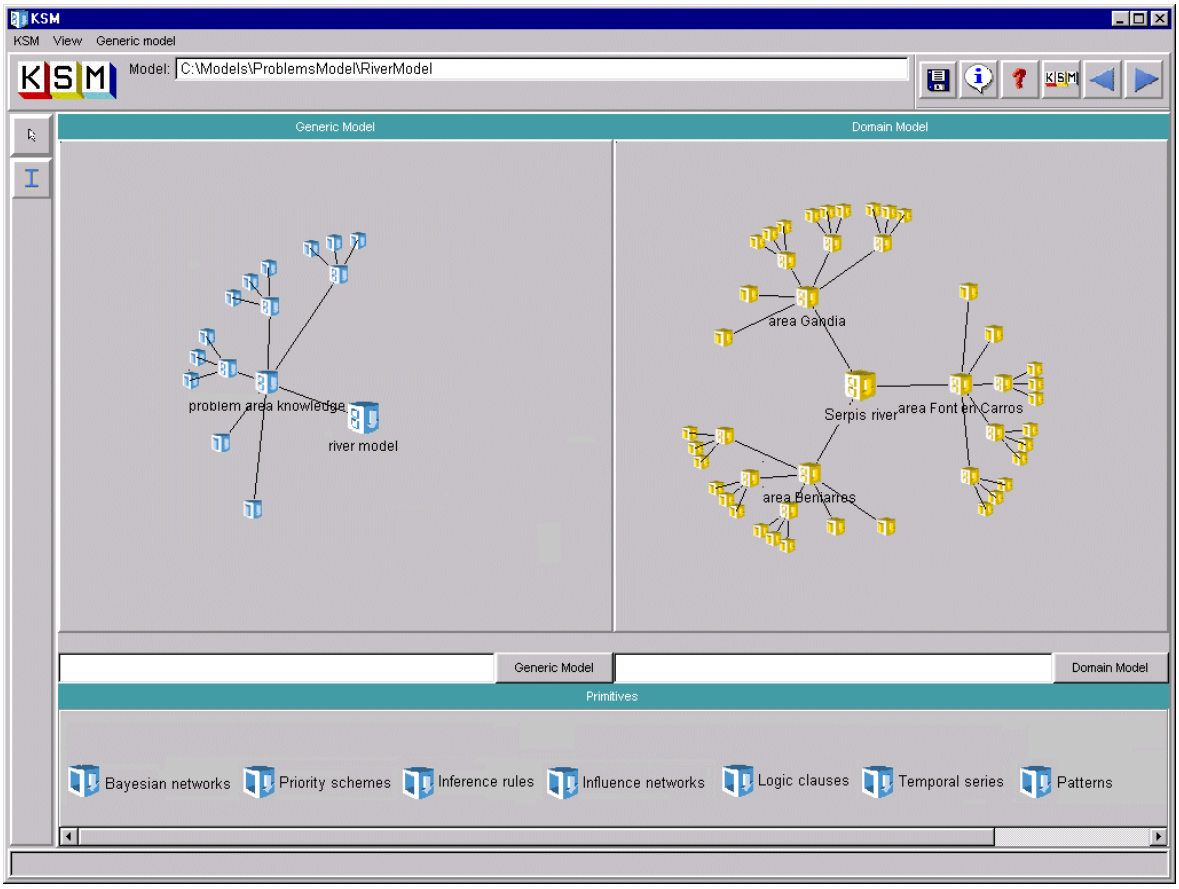

Figure 7: Screen example of the KSM environment.

In summary, the KSM software environment provides the following facilities:

a) A *user interface for knowledge modeling*, following the knowledge area paradigm. This interface consists of: (1) a graphical window-based view of knowledge modules providing visual facilities to create, modify and delete components, (2) the Link language interpreter which allows the developer to formulate high level problem-solving strategies that integrate basic components, and (3) the Concel language compiler to define common terminologies shared by different modules. Figure 7 presents a general screen presented by the KSM environment showing the knowledge areas components of a structured model.

b) A *library of reusable software components* (the primitives of representation). They may be either conventional or knowledge-based modules. Examples of general knowledge-based primitives are: rule-based primitive with forward and backward chaining inference procedures, frame-based primitive with pattern-matching

procedures, constraint-based primitive with satisfaction procedures, etc. The library is open to include new components according to the needs of new applications and they can be programmed by using different languages (C++, Prolog, etc.).

c) A *user interface for execution.* This interface allows the developer to execute knowledge models to validate them. The evaluation may be done either for the whole model or parts of it. Using the interface, the developer may select tasks to be executed, provide input data and consult results and explanations. The execution makes use of the Link interpreter to execute methods and the primitives of representation to execute the basic inferences.

3. A general approach to agent-based design using KSM

As indicated above, KSM is a knowledge modelling tool which finds use in configuring and organizing a variety of knowledge and problem-solving methods. For an agent-based system, this tool can be very convenient for modelling the knowledge of the different system components to produce a social behavior. In particular, a typical multiagent system may, in this context, include at least the following types of categories:

(1) *Local knowledge.* This encompasses problem-solving knowledge to solve the particular problems pertaining to a specific agent. In general, a knowledge-based agent is designed to perform multiple tasks. Some of these tasks may be defined to sense and to act on a particular environment by self-motivation. Each agent is capable of perceiving information about its surroundings (observables) and it is capable of performing some actions determined by applying the local problem-solving knowledge. Structure and form of this knowledge are specific for each particular individual agent, although there may be similarities between the same types of agents.

(2) *Social knowledge.* This encompasses what is known to each agent about its society in general and permanent terms. Included are: (i) interaction norms or protocols and (ii) local views about the other members of the society (single members or groups of members as sub-organizations). Usually, this requires to add to each agent a reflective knowledge layer which simulates how the agent reasons about itself and the other agents to distribute the problem-solving actions.

(3) *Multiagent organization.* These are types of agents and social structures which are organized in combination with a communication medium. For instance, one distinguishes between homogeneous and heterogeneous agents, between hierarchical and flat organizations, and other factors, such as communication between agents, the physical medium used by agents to establish communication, etc.

These components of a multiagent architecture can be supported by the KSM tool as follows. According to the original purpose of KSM, the tool naturally provides an answer about how to develop a model for the local problem-solving knowledge of each particular agent. Thus, *KSM provides a solution to organize the complex local problem-solving knowledge* by defining hierarchies of knowledge areas that encompass domain knowledge together with the corresponding problem-solving methods for specific problems.

From the point of view of social knowledge, the KSM methodology allows the developer to define knowledge areas specialized in social knowledge (i.e., interaction protocols for local agent models). Given the knowledge-based general approach for this case, *the social behavior of agents is open*, i.e., it can be written for each case and accessed by external users of the particular knowledge base. Interaction protocols are defined in terms of declarative sentences about criteria that indicate when it is necessary to solve local problems or send messages to other agents. In such a case, diagrams showing states and transitions are typical representations, where each state represents a task of a particular agent and each transition defines a particular response.

Likewise, the local view of an agent about the others can be written for each case in the particular knowledge base and can even be modified by the problem-solving experience. Also, it is important to note the usual presence of the reflective layer to allow the agent to reason about itself and to decide which actions will be carried out by other agents when its own competence or function is limited. To this effect, *KSM provides a*

meta-knowledge representation language about tasks, methods, and knowledge areas that can be used to express the specific competence or function of each agent. For instance, each agent may possess a 'self-view' defined as a set of tasks, where each task includes a set of types of premises (inputs) and types of responses (outputs). The remaining agents will usually possess a subset of these tasks in their social knowledge.

Finally, KSM can also provide support for multiagent organizations. In this context, KSM may help to define families of homogeneous agents that share a similar knowledge structure. This is supported by the idea of a generic model that identifies an abstract knowledge structure and general control mechanisms common to different agents of the same type. In addition, KSM also provides common terminologies supported by what is called 'conceptual vocabularies' that can be shared by different agents. Thus, KSM provides on this instance *a solution for knowledge-sharing and reuse* that facilitates the development of each agent within families of individual agents. Lastly, KSM may also provide a mechanism for easier communiation between different agents by selection of the most appropriate solution for each case. This can be derived from the idea of *a basic representation that locally encapsulates and controls the most effective communication medium* for each agent.

In addition to the above applications of KSM to multiagent systems, this tool also helps the developer in accomplishment of the following tasks: (1) *agent programming* - KSM provides software components (representation primitives) facilitating the development of the whole application; these software constructs are knowledge-based components that offer a high degree of adaptability derived from local declarative representations; and (2) *agent validation and maintenance* - KSM enables the developer to validate each individual module which facilitates and systematizes the entire validation process.

The next section describes how KSM provided this type of assistance in the development of two real world multiagent systems.

4. Examples of agent-based systems developed using KSM

This section describes how the KSM tool was used in the developement of two multiagent systems. The first system, called EXPERNET, was developed for the problem of distributed management of a telecommunication national network [14]. The second application, called SAIDA, was built to assist operators in emergency management in the domain of river floods [15].

4. 1. Example No 1: Distributed network management

The governement of Ukraine initiated the programme *Informatization of Ukraine*, directed towards the development of telecommunication infrastructures and information technologies at a national level, during the years 1996-2000. In this context, the goal of the EXPERNET project (funded by the European Union's Inco-Copernicus Programme) was to develop a distributed expert system to support network administrators in the management of a national data network. One of the main national networks of Ukraine, was chosen as the experimental zone of the project, that provides Internet services (email, ftp, news, telnet, access to www-servers, etc.). Figure 8 illustrates the structure of the network that includes a significant number of network nodes provided by independent organisations.

At each node of this network, there is a node administrator, responsible of maintaining a convenient quality of service. The main goals of the administrator is *fault detection, performance optimisation* (e.g., changing services to different hosts, changing routing tables etc.) and *network re-configuration* (e.g. increasing/decreasing the capacity of channels, leasing or cancelling new lines, installing new equipment etc.). Coordination problems play an essential role in the tasks of administrators (for instance, when a node lacks observables which are available to another node or when nodes with overlapping problem-solving capacities have to agree to apply a solution). Nodes may correspond to any of three levels of a hierarchy (national, regional and district) and they can

communicate in accordance with certain conversation patterns to overcome coordination problems.

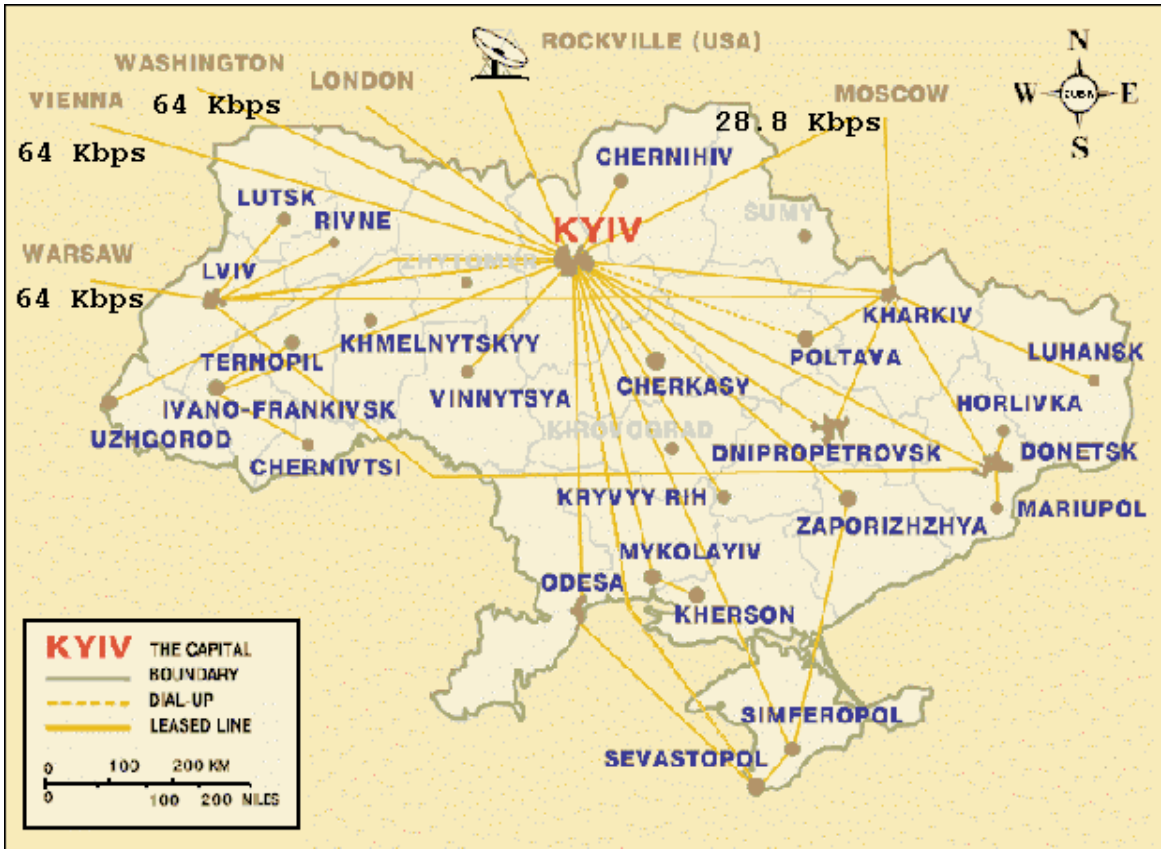

Figure 8: Structure of a telecommunication network of Ukraine.

Agent knowledge models

Within this context, the EXPERNET system was designed to assist the node administrators in the network management tasks. The inherent distribution of the problem suggested to use a multiagent system architecture, where each management node in the network is associated to one *agent*, specialised in managing the network area that the node is responsible for. Each decision support agent communicates the results of its reasoning processes to its human administrator, which is in charge of settling the management actions to be taken. The local problem-solving competence of each decision support agent follows a three step sequence: (1) *symptom detection* to watch out for symptoms of undesired network states and behaviours (e.g. a certain service –ftp, www, etc.– does not respond, a host is unreachable, over/under-utilisation of links or equipment, etc.), (2) *diagnosis*, which is done by discriminating hypotheses of different degrees of precision on the basis of network data and the result of exploratory actions to find the causes of symptoms and (3) *repair*, where a sequence of repair actions is proposed to solve the problem. In order to achieve these tasks we adapted three well-known problem-solving methods in the knowledge engineering community: a generic *data-driven heuristic classification* method [16] for symptom detection, a version of the *establish-and-refine* method for diagnosis [17], and the *hierarchical planning* method for repair [18].

From the point of view of social knowledge within each agent, we identified interactions within a conversation based on a message passing model. Every message that is exchanged during such interactions conveys a *speech act* with which the sender tries to influence the behaviour of the receiver [19]. Within conversations there are various degrees of freedom for the involved agents, as they usually may choose from several behaviour options (in the simplest case to accept or to reject a request). This accounts for the autonomy of the network administrator within the organisation. The behaviour of an administrator in a conversation (i.e. his/her choice among the different options) is not just

determined by information respecting its local situation, but also by its knowledge and experience with other nodes in the network. This knowledge is represented in the *agent models* (this type of knowledge is also referred to as *acquaintance model* [20] or *external description* [21]). An agent maintains such local agent model of all acquaintances that it interacts with (every agent is also endowed with reflective knowledge about itself).

The main social functionalities provided by EXPERNET agents are: (1) *problem interest*, checks whether the modelled agent is believed to be interested in being notified about a problem, (2) *plan interest*, checks whether the modelled agent is believed to be interested in being notified about a given plan, (3) *plan rights*, checks whether there is a need to obtain the agreement of the modelled agent for enacting a given plan, (4) *observation capability*, checks whether the modelled agent is believed to be capable of acquiring the value of a given observable, (5) *diagnosis capability*, checks whether the modelled agent is believed to be able to perform diagnosis for a given symptom, (6) *plan repair capability*, checks whether the modelled agent is believed to be able to elaborate a plan for a given problem, (7) *plan refinement capability*, checks whether the modelled agent is believed to be capable of refining a given abstract plan for a given problem.

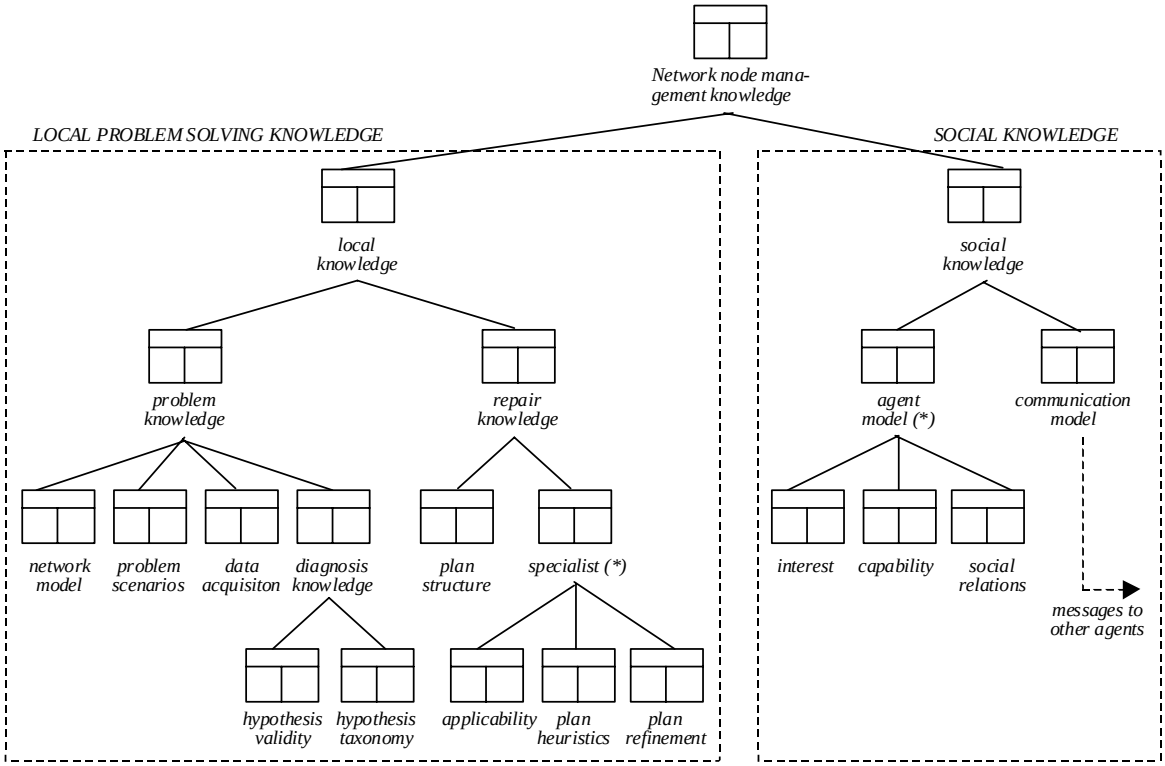

Figure 9: General structure of the knowledge of an individual agent of the EXPERNET system. This structure is described as a hierarchy of knowledge areas of different levels of aggregation and it is common for every agent. The two areas marked by (*) correspond to generic areas that normally are instanced by more than one areas at domain level (see figure 10).

All this knowledge (both local problem-solving and social knowledge) was structured and organized following the methodology supported by the KSM tool. Figure 9 shows the generic knowledge model corresponding to EXPERNET agents. This figure shows a general organization of the agent expertise in knowledge-areas that is common for every agent. In the figure, the whole knowledge model, represented by the area *network node management knowledge*, is divided into two areas corresponding respectively to the local and social knowledge. The local knowledge includes two knowledge areas: (1) *problem knowledge*, with expertise respecting undesired states of the network of the experimental zone, and (2*) repair knowledge*, with expertise of how undesired states of the network can be overcome. The problem knowledge of every agent is given by a collection

of four sub-areas: (1) *network model*, knowledge about the network entities that provides functionalities for data abstraction and symptom refinement, (2) *problem scenarios*, about network states (this area provides the support for symptom detection in the frame of a heuristic classification symptom detection method) (3) *data acquisition*, knowledge respecting exploratory action (provides a method to acquire additional observables) (4) *diagnosis knowledge*, that comprises two sub-areas, relating to hypothesis validity and hypothesis taxonomy. The structure of the repair knowledge area is influenced by the hierarchical planning method that is used for the routine design of repair plans. It comprises just two classes of knowledge areas: (1) *specialist*, the structure of the *class* of knowledge that specialists are endowed with, and (2) *plan structure*, this knowledge area models expertise concerning the structure of repair plans. It is thus capable of co-ordinating the activities if specialists during the hierarchical planning process, by decomposing and re-composing abstract plans as well as partial.

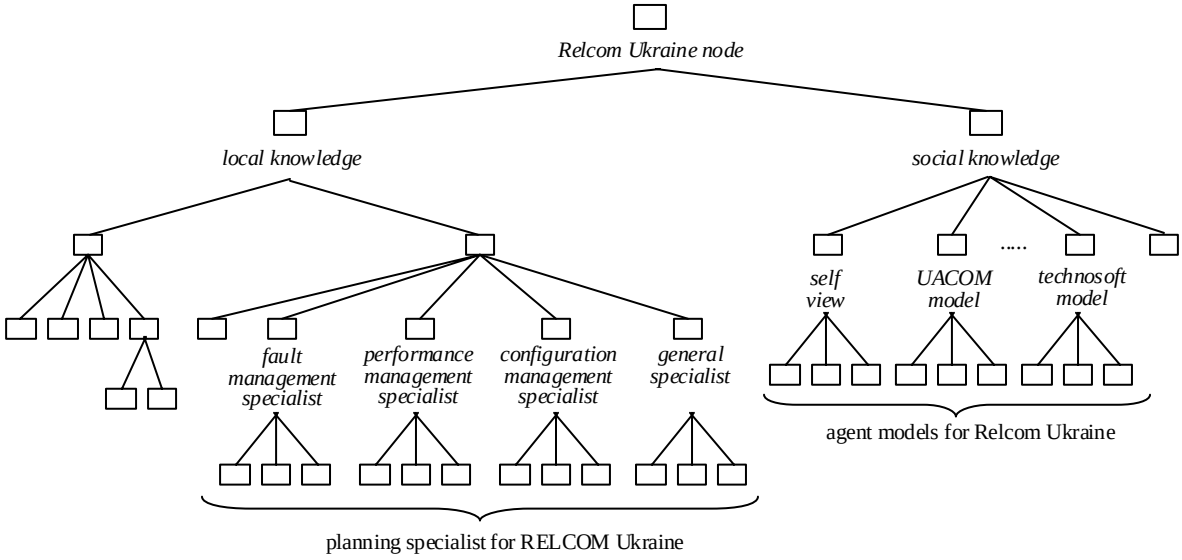

Figure 10: Structure of the knowledge model of a particular EXPERNET agent (responsible of the RELCOM Ukraine node). This model corresponds to a particularization of the generic model that figure 9 shows. The figure shows with explicit names the particular knowledge modules for agent models and planning specialists.

The *social knowledge* includes two classes of subareas: (1) *agent model*, with all the information that an agent has about an acquaintance, (2) *communication model*, this knowledge area models expertise respecting an agent's capabilities to communicate with its acquaintances (it knows about relations between an agent's desires respecting the actions of other and the speech acts that have to be sent in consequence). The agent model is subdivided in three parts, which are modelled as sub-areas: (1) the *capability* describes what the modelled agent is capable of, (2) the *interest* describes what information the modelled agent is interested in, (3) the *social relation* complies knowledge respecting the authority, relations between agents, etc.

At the implementation level, the previous generic knowledge structure is supported by several primitives of representation that provide representation languages together with inference procedures. In this particular multiagent model, there is a primitive of representation that was used to communicate agents. This primitive supports the area *communication model* (within the social knowledge) and corresponds to a software component that encapsulates the communication mechanisms between agents. Thus, when a particular agent sends a message to another agent, this is done by using a procedure of this primitive that controls the specific communication protocol according to the type of message.

The generic model (together with the primitives of representation) is shared to build each particular agent knowledge model. For instance, figure 10 shows the particular

structure of the knowledge model corresponding the agent called Relcom Ukraine node (the decision support agent at the node of Relcom Ukraine). This agent shares the generic structure of EXPERNET agents (local and social knowledge, etc.). The figure makes explicit the set of knowledge areas corresponding to planning specialists and agent models. Within agent models, note that there is a particular set of areas corresponding to the self view of the Relcom Ukraine agent that provide the required reflective capability for this agents, besides the local views of the other agents (UACOM, Technosoft, etc.).

4.2. Example No. 2: Emergency management

As a second example of a multiagent system where KSM was applied, this section describes the SAIDA system for emergency management. This system belongs to a national programme in Spain (SAIH, Spanish acronym for Automatic System of Hydrology Information) whose goal is to install an advanced information infrastructure in the main river basins. This programme includes the use of new technologies for data sensoring and communications to get on real time the information on rainfall, water levels and flows in river channels.

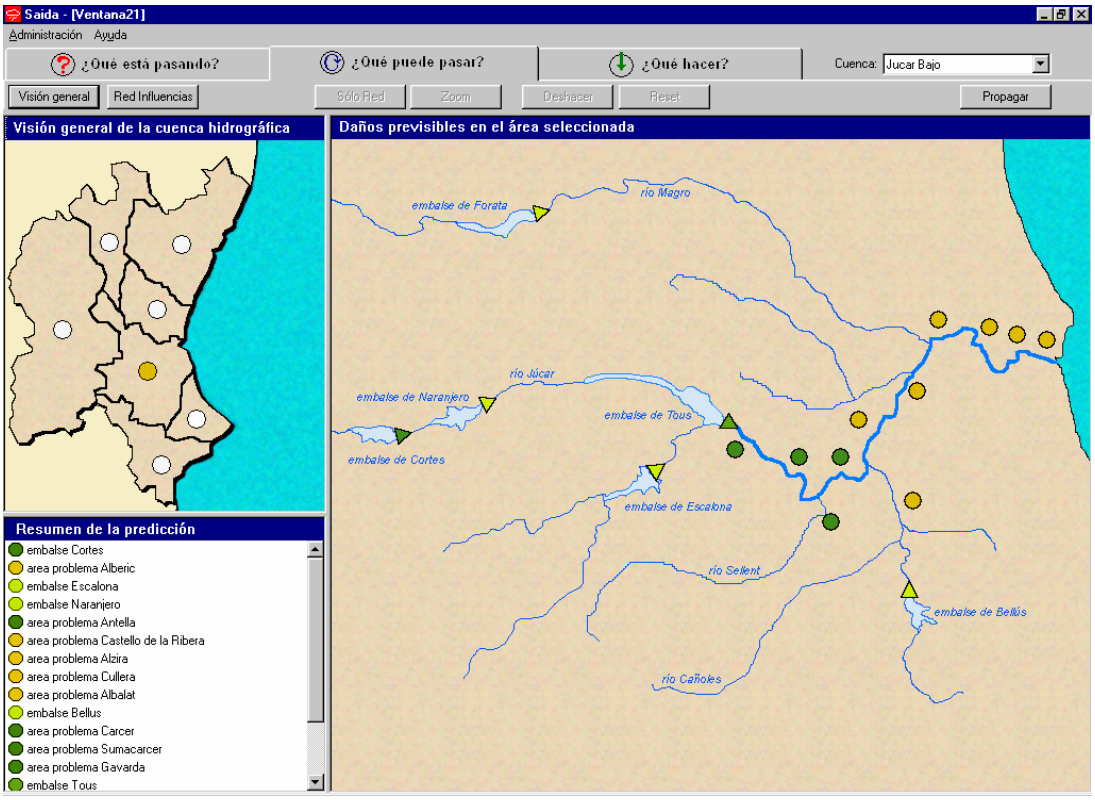

Figure 11: Screen example of the user interface presented by the SAIDA system.

One of the main purposes of this information system is to help in decision making during flood emergency situation. However, receiving a large amount of detailed data flow (every 5 minutes data of about hundreds of sensors in a typical watershed) requires an intelligent interface able to translate the data flow to a conceptual framework close enough to the natural intuitions followed by the persons in charge of control. To meet this objective, the SAIDA system was developed with the following goals: (1) to identify relevant problematic events to be detected, (2) to predict the future behaviour assuming that the current state of control is maintained, (3) to recommend possible plans of control actions on the causes of the detected problems and, (4) to predict what will happen if the recommended plans or some variants for them are applied.

Given that the SAIH programme will develop several of such systems it was important to consider the reusability of the software architecture. Also, given the incremental installation of the information ifrastructure at different river basins, it is

impossible to start with closed versions of the systems because data available about the physical features of the river and reservoir systems are very unequally distributed. This leads to use an open structure based on a knowledge based architecture where the results of the experience using the system can be applied to refine the knowledge contents. Autonomy of the models was also required to ensure a good maintenance policy for extension of the system. All these circumstances lead to an intelligent, knowledge based agent architecture where the main functions of problem detection, reservoir management, water resources behavior and civil protection resources management are encapsulated in specialized agents integrated by relations of physical behavior and multiplan generation for flood problems management.

Agent knowledge models

In this problem, according to the different nature of the goals to achieve, the following types of agents were considered (figure 12):

- *Hydraulic agents* that are responsible to give answers about the behavior of the physical phenomena (the rainfall, the runoff produced by the rainfall incidence in the land, the concentration of the runoff in the main river channel, the reservoir operation and the flow in the lower levels on the river).
- *Problem detection agents*, responsible of evaluating the flood risk in a particular geographical area.
- *Reservoir management agents*, which embody criteria for exploitation strategy for each reservoir.
- *Civil protection agents*, responsible to provide with resources of different types according to the demands of the problem detection agents.

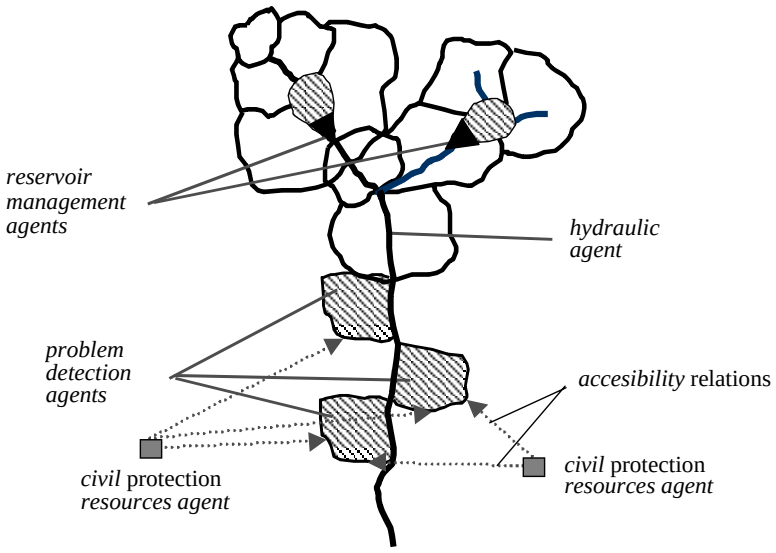

Figure 12: Types of agents in the SAIDA system for a river basin.

Local problem detection agents are responsible of detecting and predicting potential problems at specific locations of the river basin. For this purpose they receive input data from sensors, analyses them and, when a particular agent identifies a potential dangerous scenario, it asks for a prediction of behaviour to the corresponding hydraulic agents. Once, the problem detection agent receives the prediction from hydraulic agents, it interprets the information to conclude the level of severity of the future problem. When local problem detection agents predict the existence of future problems they ask for limiting the water flow upstream their areas location. In general, different problem detection agents (or even other reservoir management agents) may ask for a limitation to the same reservoir, so the reservoirs must adapt their discharge policy in order to avoid several problems and, at the same time, to maintain its own risk under a reasonable level. To deal with the case of multiple reservoirs, it is required to keep an homogeneous risk level among all the

reservoir management agents so an individual agent must accept to increase its risk level if the other *cooperating agents* accept also to increase. Thus, the global strategy to attain this solution is based on a method that increases or decreases step by step the risk level of reservoirs, following the social rule that all the reservoirs must have similar risk levels.

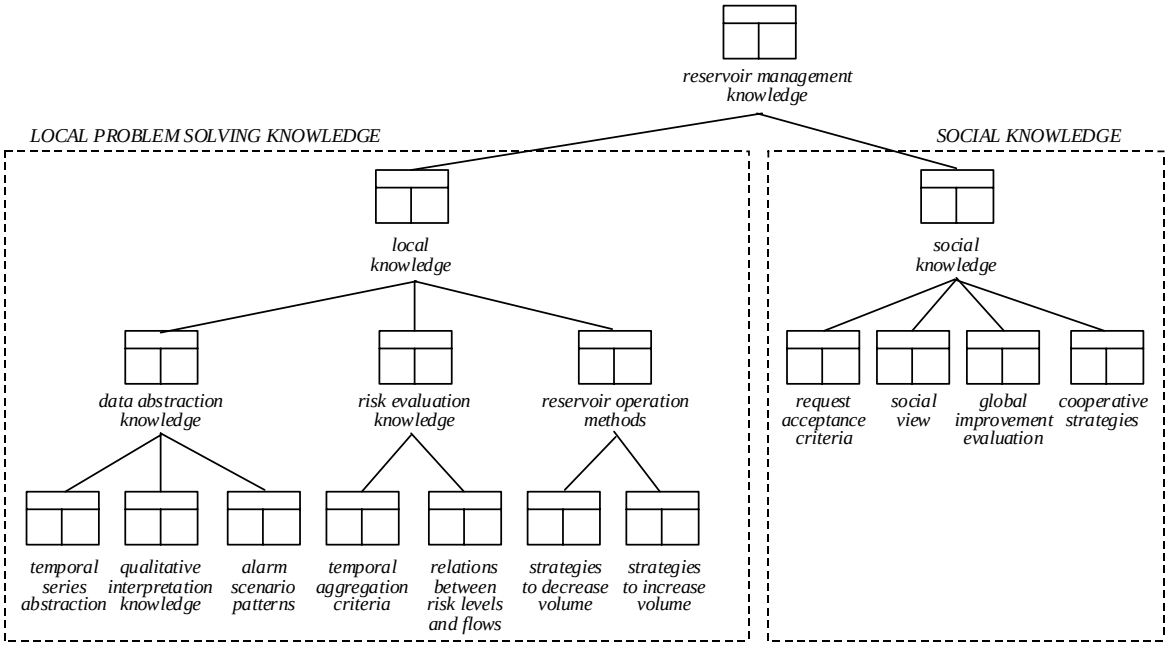

Figure 13: Example of generic structure of the knowledge for a type of agent of the SAIDA system (the reservoir management agent).

Each type of agent has its local knowledge model that includes both local and social knowledge. For illustration purposes, figure 13 shows part of the abstract knowledge model defined for the one of the main agents of this model, the *reservoir management agent*. Each river includes several instances of such a model depending on the number of reservoirs in that river (e.g., the Jucar river includes six instances). The reservoir management agent is responsible of suggesting local control actions related to the operation of a single reservoir, directed to increase or to decrease the water discharge. For this purpose the agent perceives directly the state of the river by using sensor data from the information system but also this agent interacts with other agents (e.g., problem detection agents or other reservoir management agents) that request specific control actions to modify the discharge. The generic structure of figure 13 includes a top-level area that represents the whole model, the *reservoir management knowledge* area, that is divided into the local problem-solving knowledge and the social knowledge. The local problem-solving knowledge is divided into three areas: (1) *data abstraction knowledge* to interpret and abstract sensor data in order to classify the severity of the current situation, (2) the *risk evaluation knowledge*, that includes expertise about how to evaluate the future behaviour of the reservoir in order to estimate the risk level, and (3) *reservoir operation methods*, that includes control strategies to increase or decrease the volume. The social knowledge includes four areas: (1) *request acceptance criteria*, that encapsulates the criteria to decide how to answer to the request from other agents (problem detection agents or reservoir management agents), (2) *social view*, that includes the set of agents with which the reservoir management agent is related (e.g., the corresponding hydraulic agents related to this agent, etc.), (3) *global improvement evaluation criteria*, to evaluate different risk levels of several agents in order to know if a particular control action contributes to improve the global situation, and (4) *cooperative strategies*, that are used to determine the best control action (e.g., in order to decrease the risk level of the reservoir, this area provides criteria to discriminate between two different control actions: either to ask upper reservoirs to decrease their flow or to increase locally its discharge).

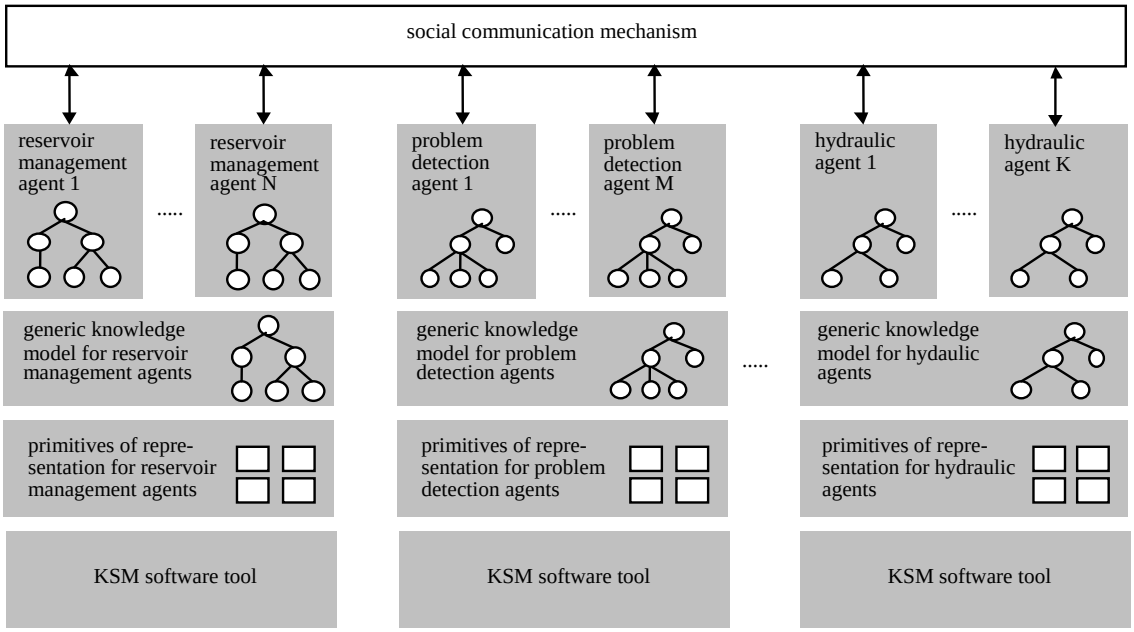

Figure 14: Distributed knowledge model using KSM to support the multiagent architecture of the SAIDA system.

As a main difference from the point of view of knowledge organization between SAIDA and EXPERNET systems, the SAIDA system includes several types of agents with different knowledge structures (whereas EXPERNET only includes one type of agent), which makes more complex the final software architecture. Figure 14 shows a summary of the final solution adopted by the SAIDA system, where efficiency was an important factor. In this architecture, there are several copies of the KSM tool, where each one supports a family of agents with the same generic knowledge structure. For example, the left hand side of the figure shows the solution for reservoir management agents. Here, a copy of the KSM tool serves as a software platform where it is installed a library of primitives of representation (reusable software components) that are used to implement the generic knowledge model for reservoir management agents (see figure 13). In its turn, this structure of generic model is shared by the particular knowledge models of each reservoir management agent that includes specific knowledge bases. This organization is similar for the other types of SAIDA agents: problem detection agents, civil protection agents and hydraulic agents. In addition to that, a global mechanism is used to communicate agents according to the required individual autonomy in the model. Thus, here, KSM also gives a solution to build general knowledge structures that are reused and shared by different instances of agents, giving the required freedom to write the particular adaptations of each agent in their particular knowledge bases.

Representation Language	**Characteristics**
Inference rules	A rule-based primitive with backward/forward chaining inference procedures
Frames	A frame-based primitive with fuzzy pattern-matching inference methods.
Logic clauses	A logic-based primitive with inference procedures based on automatic deduction
Belief networks	A network-based primitive with bayesian inference procedures.
Influence networks	A network-based primitive to represent functionally a dynamic system.
Temporal series functions	A function-based primitive to perform operations on termporal series.

Figure 15: Main reusable knowledge-based software components (primitives of representation) provided by KSM to develop the SAIDA system.

From the point of view of software programming, KSM provided particular reusable software components specialized in knowledge representation (the primitives of representation) that helped in building the particular models. In the case of SAIDA, figure 15 shows the main set of general primitives used. This set includes different knowledge representation languages such as rules, frames or belief networks that were used to develop local knowledge bases within agents.

4.3. Summary

This section summarizes how KSM was used for the EXPERNET and SAIDA case studies described above. In regard to the development of local problem-solving models for EXPERNET, KSM provided support in defining a generic abstract knowledge structure, common to all specific agents, in terms of a set of 14 knowledge areas and 9 types of knowledge bases. In the SAIDA system, KSM provided four types of knowledge structures, one for each type of agent (hydraulic agent, problem-detection agent, reservoir management agent, and civil protection agent). There were 44 knowledge areas with a set of 33 types of knowledge bases. In both case studies, KSM provided solutions for configuring and organizing the local problem-solving knowledge of each type of agent.

In regard to the development of social models for the EXPERNET system, KSM was useful to define the specific social knowledge from which agent interaction could be derived. This was done with 5 knowledge areas and 4 types of knowledge bases pertaining to communication and agent models, whereby one of the latter included a reflective model for self-view. In the SAIDA system, KSM likewise provided a way to implement social knowledge. For instance, 5 knowledge areas were used in the case of reservoir management to determine the social interactions among cooperative strategies. In both case studies, thanks to the use of symbolic models for each agent, it was possible to specify the particular details for each agent. For example, in the case of a reservoir management agent, particular criteria could be established to compare its inherent risk level to those of the remaining agents so that cooperative action procedures could be accepted or rejected.

Finally, regarding a multiagent organization of the EXPERNET system, KSM was applied to formulate specific agents. In the final implementation, there were three complete agents (corresponding to the three nodes Rtelcom, Ukraine, UACOM, and Technosoft) which resulted in 39 knowledge bases (27 for local knowledge and 12 for social knowledge). All of them shared the same abstract knowledge structure. Here, KSM provided a solution to reuse the abstract structure together with the corresponding software components. Communication between agents was solved by a knowledge area supported by a primitive of representation capable of sending messages via a communication network and awaiting responses according to the specific protocol for each agent. In this case, KSM provided a solution for encapsulating and integrating this module into the entire architecture from a homogeneous knowledge-based perspective. In addition to that, one of the last implementations of the SAIDA case study was applied to a Spanish watershed involving two rivers (Jucar and Serpis), with a total of 26 agents. In this set, 7 agents corresponded to reservoir management agents. These data show the importance of knowledge-sharing and reuse capabilities to minimize development efforts and to facilitate validation and maintenance of the final system.

5. Conclusions

As shown by the ideas outlined in this chapter, the kind of support available from knowledge modelling tools for the development of agent-based systems is particularly significant in terms of the following capabilities:

- *Knowledge configuring and encapsulation facilities* to formulate a model about the cognitive capacities of each individual agent. Knowledge modelling tools can provide a solution, using descriptions based on natural intuitions, to separate and organize the various knowledge categories in a model.
- *Meta-knowledge description entities* facilitate formulation and inclusion of reflective layers that normally would be required to give each agent the capability of reasoning about itself as well as about the other agents (acquaintance models).

- *Knowledge-sharing and reuse support* based on the possibility of developing abstract knowledge structures that include general problem-solving knowledge together with global structuring patterns. These structures can be reused for the development of different particular agents sharing the same knowledge structure and, possibly, contents of knowledge bases.
- *Open architectures* in which - in addition to local problem solving knowledge corresponding to the local capability of an individual agent - it is possible to formulate explicitly for each agent social norms for interaction mechanisms, using particular symbolic declarative representations. This facilitates revision and adaptation for comprehension and maintenance purposes.
- *Programming facilities* based on the management of libraries containing reusable pre-programmed knowledge-based software components together with a natural combination scheme to implement the target system. Availability of this type of componnts can significantly decrease the programming efforts required for development of a global system.

All of these factors were particularly handled by the KSM environment as shown for the development of the multiagent systems of the EXPERNET and SAIDA case studies, where eficient solutions were provided for integration of agent-based and knowledge-based technologies. This combination leads to consideration of a next generation of tools defined as software frameworks, where knowledge modelling tools are enhanced by agent-based features to provide a whole and complete environment that may enable developers to formulate, implement, and maintain large-scale applications developed by reusable software components and knowledge structures from a global agent-based view. For example, within these agent-based features, it will be necessary to consider factors, such as: (1) computational inter-communication-distributed mechanisms integrated within these frameworks, (2) representation languages and open tools to give support to a wide range of different social decentralized interaction methods among agents, and (3) flexible abstract agent models and interaction mechanisms that can be reused in particular applications.

References

[1] M. Molina: “Desarrollo de Aplicaciones a Nivel Cognitivo Mediante Entornos de Conocimiento Estructurado” PhD Thesis, Technical University of Madrid. November, 1993.

[2] J. Cuena, M. Molina: “KSM: An Environment for Knowledge Oriented Design of Applications Using Structured Knowledge Architectures” in *Applications and Impacts. Information Processing’94*, Volume 2. K. Brunnstein y E. Raubold (eds.) Elsevier Science B.V. (North-Holland), 1994 IFIP.

[3] J. Cuena, M. Molina: "The Role of Knowledge Modeling Techniques in Software Development: A General Approach Based on a Knowledge Management Tool". *International Journal of Human-Computer Studies* (2000) 52, 385-421.

[4] M. Molina, J.L. Sierra, J.M. Serrano: "A Language to Formalize and to Operationalize Problem Solving Strategies of Structured Knowledge Models" *8th Workshop on Knowledge Engineering: Methods and Languages KEML 98*. Karlsruhe, Alemania, 1998.

[5] M. Molina, J. Hernández, J. Cuena: "A Structure of Problem Solving Methods for Real-time Decision Support in Traffic Control*" International Journal of Human and Computer Studies* (1998) 49.

[6] A. Newell: "The Knowledge Level" in *Artificial Intelligence* Vol 18 pp 87-127, 1982.

[7] B.J. Wielinga, A.T. Schreiber, J.A. Breuker: "KADS: A Modelling Approach to Knowledge Engineering". *Knowledge Acquisition*, 1992.

[8] A.R. Puerta, S.W. Tu, M.A. Musen: “Modelling Tasks with Mechanisms”. *International Journal of Intelligent Systems, Vol. 8*, 1993.

[9] B. Chandrasekaran, B.: “Towards a Taxonomy of Problem Solving Types” *A.I. Magazine* 4 (1) 9-17, 1983.

[10] B. Chandrasekaran: "Generic Tasks in Knowledge Based Reasoning: High Level Building Blocks for Expert Systems Design" *IEEE Expert*, 1986.

[11] L. Steels: "Components of Expertise" *AI Magazine*, Vol. 11(2) 29-49.

[12] J. McDermott: “Preliminary Steps Toward a Taxonomy of Problem Solving Methods” in *Automating Knowledge Acquisition for Expert Systems*, S.Marcus ed., Kluwer Academic, Boston, 1988.

[13] M. Molina, J. Sierra, J. Cuena: "Reusable Knowledge-based Components for Building Software applications: A Knowledge Modelling Approach*" International Journal of Software Engineering and Knowledge Engineering*, Vol. 9 No. 3 (1999) 297-317.

[14] M. Molina, S. Ossowski: "Knowledge Modelling in Multiagent Systems: The Case of the Management of a National Network" in "Intelligence in Service and Networks. Paving the Way for an Open Service Market*" Lecture Notes in Computer Science 1597*, Springer, 1999.

[15] J. Cuena, M. Molina: "A Multiagent System for Emergency Management in Floods" in "Multiple Approaches to Intelligent Systems*", Lecture Notes in Artificial Intelligence 1611*, Springer, 1999.

[16] W. Clancey: “Heuristic Classification”. *Artificial Intelligence 27*, 1985.

[17] B. Chandrasekaran, T. Johnson, J. Smith: “Task-Structure Analysis for Knowledge Modelling”. *Communications of the ACM 35 (9)*, 1992.

[18] D. Brown, B. Chandrasekaran: "Design Problem-solving: Knowledge Structures and Control Strategies", Morgan Kaufman, 1989.

[19] H.J. Müller: “Negotiation Principles” in: *Foundations of DAI* (O’Hare & Jennings, eds.), Wiley, 1996.

[20] D. Cockburn, N. Jennings: “ARCHON: A Distributed Artificial Intelligence System for Industrial Applications” in: *Foundations of DAI* (O’Hare & Jennings, eds.), Wiley, 1996.

[21] J. Sichman, Y. Demazeau: “Exploiting Social Reasoning to deal with Agency Level Inconsistencies”. *Proc. ECAI-94*, 1994